**CS 6043/5143: Computer Networking**

**FALL 2019**

**PROJECT 3**

**Given: Nov. 11, 2019**

**Due: Nov. 25 (Monday), 2019 (NO LATER THAN 11:59PM)**

**Submission Instructions:**

1. Submit only on-line files on Blackboard before midnight. No hard copy will be accepted.

2. Wireshark files for this project can be found in the zip file “Project\_3\_Wireshark\_Traces\_Fall\_2019.zip”.

**Total possible points: 10**

**Part I: IP**

Load the file ‘IP\_project\_3.pcapng’ in Wireshark and answer the following questions. The trace was generated while executing three *traceroute* commands with three different UDP datagram sizes (56, 2000, and 3000 bytes) on a computer with IP 10.63.7.60. The IP address of the target host was 103.74.84.13. Include screenshots in each case.

* 1. (0.5 pts) Select any of the ICMP echo packets from the trace. Within the IPv4 packet header, what is the value in the upper layer protocol field?
  2. (1.0 pts) Select any of the UDP packets generated from the second *traceroute* command. How many bytes are in the IP header? How many bytes are in the payload of the IP datagram? Explain how you determined the number of payload bytes.
  3. (1.0 pts) Observe the UDP packets generated from the first *traceroute* command. Which fields in the IP datagram always change from one datagram to the next within this series of messages sent by 10.63.7.60?
  4. (0.5 pts) What are the values in the Identification field and the TTL field of the first ICMP packet (ping reply) received by 10.63.7.60?
  5. (1.0 pts) Do the values of the Identification field and the TTL field remain unchanged for all of the ICMP TTL-exceeded replies sent to your computer (IP 10.63.7.60) by the nearest (first hop) router? Why?

**Part II: DHCP**

Load the file ‘DHCP\_project\_3.pcapng’ in Wireshark and answer the following questions. The trace was generated while releasing the host’s current IP address and renewing it multiple times (Release → Renew → Renew → Release → Renew). Include screenshots in each case.

[Hint: Apply display filter “bootp” in Wireshark to filter out DHCP packets]

* 1. (0.5 pts) What is the link-layer address and the hardware address length of the host?
  2. (0.5 pts) What IP address is the DHCP server offering to your host in the first DHCP Offer message?
  3. (1.0 pts) Explain the purpose of the lease time. How long is the lease time in your trace?
  4. (1.0 pts) Clear the *bootp* display filter. Were any ARP packets sent or received in the trace? If so, explain the purpose of those ARP packets with a screenshotted example.

(3.0 pts) **Part III: Socket Programming – ICMP Ping**

Ping is a computer network application used to test whether a particular host is reachable across an IP network. It is also used to self-test the network interface card of the computer or as a latency test. It works by sending ICMP “echo reply” packets to the target host and listening for ICMP “echo reply” replies. The "echo reply" is sometimes called a pong. Ping measures the round-trip time, records packet loss, and prints a statistical summary of the echo reply packets received (the minimum, maximum, and the mean of the round-trip times and in some versions the standard deviation of the mean).

Your task is to develop your own Ping application in Python. Your application will use ICMP but, in order to keep it simple, will not exactly follow the official specification in RFC 1739. Note that you will only need to write the client side of the program, as the functionality needed on the server side is built into almost all operating systems.

You should complete the Ping application so that it sends ping requests to a specified host separated by approximately one second. Each message contains a payload of data that includes a timestamp. After sending each packet, the application waits up to one second to receive a reply. If one second goes by without a reply from the server, then the client assumes that either the ping packet or the pong packet was lost in the network (or that the server is down).

**What to Hand in**

*You will hand in the complete client code and screenshots of your Pinger output for four target hosts, each on a different continent.*

**Code**

Below you will find the skeleton code for the client. You are to complete the skeleton code. The places where you need to fill in code are marked with #Fill in start and #Fill in end. Each place may require one or more lines of code.

**Additional Notes**

1. In “receiveOnePing” method, you need to receive the structure ICMP\_ECHO\_REPLY and fetch the information you need, such as checksum, sequence number, time to live (TTL), etc. Study the “sendOnePing” method before trying to complete the “receiveOnePing” method.
2. You do not need to be concerned about the checksum, as it is already given in the code.
3. This lab requires the use of raw sockets. In some operating systems, you may need administrator/root privileges to be able to run your Pinger program.

**Testing the Pinger**

* First, test your client by sending packets to localhost, that is, 127.0.0.1.
* Then, you should see how your Pinger application communicates across the network by pinging servers in different continents.

**Skeleton Python Code for the ICMP Pinger**

from socket import \*

import os

import sys

import struct

import time

import select

import binascii

ICMP\_ECHO\_REQUEST = 8

def checksum(string):

csum = 0

countTo = (len(string) // 2) \* 2

count = 0

while count < countTo:

thisVal = ord(string[count+1]) \* 256 + ord(string[count])

csum = csum + thisVal

csum = csum & 0xffffffff

count = count + 2

if countTo < len(string):

csum = csum + ord(string[len(string) - 1])

csum = csum & 0xffffffff

csum = (csum >> 16) + (csum & 0xffff)

csum = csum + (csum >> 16)

answer = ~csum

answer = answer & 0xffff

answer = answer >> 8 | (answer << 8 & 0xff00)

return answer

def receiveOnePing(mySocket, ID, timeout, destAddr):

timeLeft = timeout

while 1:

startedSelect = time.time()

whatReady = select.select([mySocket], [], [], timeLeft)

howLongInSelect = (time.time() - startedSelect)

if whatReady[0] == []: # Timeout

return "Request timed out."

timeReceived = time.time()

recPacket, addr = mySocket.recvfrom(1024)

#Fill in start

#Fetch the ICMP header from the IP packet

#Fill in end

timeLeft = timeLeft - howLongInSelect

if timeLeft <= 0:

return "Request timed out."

def sendOnePing(mySocket, destAddr, ID):

# Header is type (8), code (8), checksum (16), id (16), sequence (16)

myChecksum = 0

# Make a dummy header with a 0 checksum

# struct -- Interpret strings as packed binary data

header = struct.pack("bbHHh", ICMP\_ECHO\_REQUEST, 0, myChecksum, ID, 1)

data = struct.pack("d", time.time())

# Calculate the checksum on the data and the dummy header.

myChecksum = checksum(str(header + data))

# Get the right checksum, and put in the header

if sys.platform == 'darwin':

# Convert 16-bit integers from host to network byte order

myChecksum = htons(myChecksum) & 0xffff

else:

myChecksum = htons(myChecksum)

header = struct.pack("bbHHh", ICMP\_ECHO\_REQUEST, 0, myChecksum, ID, 1)

packet = header + data

mySocket.sendto(packet, (destAddr, 1)) # AF\_INET address must be tuple, not str

# Both LISTS and TUPLES consist of a number of objects

# which can be referenced by their position number within the object.

def doOnePing(destAddr, timeout):

icmp = getprotobyname("icmp")

# SOCK\_RAW is a powerful socket type. For more details: http://sock-raw.org/papers/sock\_raw

mySocket = socket(AF\_INET, SOCK\_RAW, icmp)

myID = os.getpid() & 0xFFFF # Return the current process i

sendOnePing(mySocket, destAddr, myID)

delay = receiveOnePing(mySocket, myID, timeout, destAddr)

mySocket.close()

return delay

def ping(host, timeout=1):

# timeout=1 means: If one second goes by without a reply from the server,

# the client assumes that either the client's ping or the server's pong is lost

dest = gethostbyname(host)

print("Pinging " + dest + " using Python:")

print("")

# Send ping requests to a server separated by approximately one second

while 1 :

delay = doOnePing(dest, timeout)

print(delay)

time.sleep(1)# one second

return delay

ping("google.com")